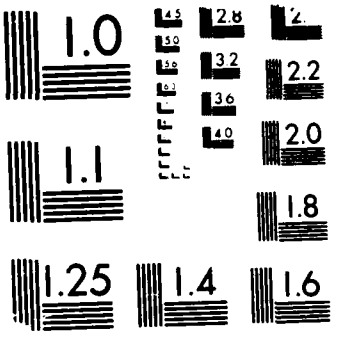
AD-A191 661 ALERTING EXPLANATION AND KNOWLEDGE ACQUISITION FOR A NAVAL THREAT ASSESSMENT EXPERT SYSTEM(U) NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA L E GADBOIS APR 88 1/1

UNCLASSIFIED NOSC/TR-1205 F/G 12/5 NL




MICROCOPY RESOLUTION TEST CHART

NOSC TR 1205

NAVAL OCEAN SYSTEMS CENTER San Diego, California 92152-5000

NOSC

Technical Report 1205
April 1988

# Alerting, Explanation, and Knowledge Acquisition for a Naval Threat Assessment Expert System

L. E. Gadbois





# NAVAL OCEAN SYSTEMS CENTER
## San Diego, California 92152-5000

E. G. SCHWEIZER, CAPT, USN
**Commander**

R. M. HILLYER
**Technical Director**

## ADMINISTRATIVE INFORMATION

The work reported here was performed by members of the Artificial Intelligence Technology Branch at NOSC, and the Computer Science Department at Carnegie-Mellon University. Research funding was provided by DARPA.

Released by
Don Eddington, Head
Artificial Intelligence
Technology Branch

Under authority of
W.T. Rasmussen, Head
Advanced $C^2$ Technologies
Division

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | Approved for public release; distribution is unlimited |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>NOSC TR 1205 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Ocean Systems Center<br>6b. OFFICE SYMBOL (if applicable)<br>Code 444 | 7a. NAME OF MONITORING ORGANIZATION |
| 6c. ADDRESS (City, State and ZIP Code)<br>San Diego, CA 92152-5000 | 7b. ADDRESS (City, State and ZIP Code) |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>Defense Advanced Research Projects Agency<br>8b. OFFICE SYMBOL (if applicable)<br>DARPA | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State and ZIP Code)<br>DARPA-Naval Technology Office<br>1400 Wilson Blvd.<br>Arlington, VA 22209 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO.: 62301E<br>PROJECT NO.: DARPA<br>TASK NO.: DARPA<br>AGENCY ACCESSION NO.: 440-CD50 |

11. TITLE (include Security Classification)

Alerting, Explanation, and Knowledge Acquisition for a Naval Threat Assessment Expert System

12. PERSONAL AUTHOR(S)

Laurence E. Gadbois

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Final | FROM Oct 1987 TO Dec 1987 | April 1988 | 28 |

16 SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | |
| | | | Artificial intelligence, explanation, alerts |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A threat assessment expert system is written which provides textual explanation of its conclusion, allows the user to modify--at run-time--the threat assessment tasks performed, and captures these user modifications for knowledge acquisition. The run-time method used to define the constituent elements of a threat situation is outlined in depth. The computer program compares the components of a threat event (defined by a knowledgeable user) with the dynamic track data base. If the threat event occurs, a warning is issued to the user. This definition of a threat event is added to the program and thus acts as run-time knowledge acquisition. The matching between the user defined attributes and the corresponding track data is displayed for explanation.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS | UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL<br>Laurence E. Gadbois | 22b. TELEPHONE (include Area Code)<br>(619) 553-4141<br>22c. OFFICE SYMBOL<br>Code 444 |

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

## EXECUTIVE SUMMARY

### PROBLEM

Threat assessment for a naval command ship is a dynamic and complex task which must evolve in response both to the external situation and to the varied and temporal needs of different users. A computer program was needed to assist the tactical watch officer with this task. For this threat assessment program to be useful, it had to be tailorable by the user to perform analyses and present information appropriate to the types of tasks the user is interested in. This tailoring must also be amenable to modification during run time in order to adapt to changing threat assessment needs.

### OBJECTIVE

The project's objective was to write a computer program to provide intelligent assistance to the tactical watch officer. This report describes a software architecture which provides three functional capabilities to the computer program: (1) to provide the user with the ability to modify, at run time, the threat assessment tasks performed by the program; (2) to explain conclusions derived by the expert system; and (3) to incorporate into the program an expert user's input of the components of tactically significant events, thereby performing knowledge acquisition.

### APPROACH

The naval threat assessment realm appears suitable for the application of expert system programming techniques. These techniques facilitate encapsulating an expert's knowledge in computer code and capitalizing on the computational power of a computer to handle the extensive information processing resulting from a dynamic data stream.

Four major points are addressed in this report: First, the foci of threat assessment personnel are on a variety of different tasks. For example, some items of high interest to ASW (antisubmarine warfare) personnel may be of marginal interest to air-strike personnel. Second, items of interest change over time in response to the current threats or exercise. These two points dictate that a useful program must be tailorable by the user to perform analyses and present information appropriate to this user at the time. Tailoring must also be amenable to modification during run time in order to adapt to changing threat-assessment needs. The third major point is that the program must be able to justify its conclusions. This is needed by the experts and programmers to validate knowledge and either to confirm to the user the validity of the conclusions or to reveal the faulty line of reasoning or invalid data input that resulted in an invalid conclusion. Fourth, the method used to tailor the program extracts components of

tactically significant events from the user. Capturing this knowledge is a way of achieving knowledge acquisition.

## RESULTS

The technique implemented in the expert system requires a user or expert to define the constituent elements of a tactically significant event which would be of interest to the watch officer. The expert system then compares the static and dynamic track data, and intelligent conclusions it has drawn using its preexisting tactical knowledge, with the conditions specified by the user. If all the threat components exist, a warning is issued to the user.

This method of having the expert user define the threat situations fulfills the need for knowledge acquistion. The pairing of user-defined threat components and the corresponding track data forms the foundation for an explanation of the warnings.

## CONCLUSIONS

The expert system fulfilled the objectives of flexibility and an organized method of explanation. It provided a crude but functional method for knowledge acquisition.

## RECOMMENDATIONS

The process for employing user definition of tactical events is good as far as it goes, but needs enhancements to allow the user greater flexibility in defining track characteristics. Not all relevant threat events are track related, so the user should be allowed to include these types of events in his definition of a threat situation. The software described in this report does not allow for the inclusion of nontrack-related conditions.

Some method for dynamically modifying the tasks to be performed should be included in any expert system which must adapt to the information needs of different users. This report describes one such method.

## CONTENTS

## NOMENCLATURE

| | |
|---|---|
| **alert** | A structure built by the user that specifies a number of tracks and a number of attributes for each of the tracks. |
| **alert condition** | One of the attributes that a track in an alert must possess. |
| **assertion** | Conclusions based directly on factural data such as that derived from data bases, or conclusions derived by applying knowledge contained in rules to factual data. |
| **category** | A classification into one of the following: subsurface, surface, air, space. |
| **CAT** | Command Action Team. An expert system written primarily in OPS83 to perform threat assessment for a naval command ship. |
| **Comp_type** | Comparison type. Can be null, within object, or between object. |
| **Comp_node** | For between-object comparisons, this number indicates the address of the alert condition to use for comparison. |
| **cpa** | Closest point of approach. Specified by a position, range, and date/time. |
| **explanation** | Textual description of why a warning was produced. |
| **expert system** | Computer program that performs an intelligent analysis of a situation. The process mimics the decisions an expert in that field would perform under the same situation. |
| **inference engine** | Determines the rules that are relevant given the current working memory, chooses which rule to fire, and executes it. |
| **knowledge** | The ability to draw conclusions based on an initial set of information. In expert systems knowledge is contained in structures called rules and the initial set of information needed by the rule is called the antecedent. |
| **LHS** | Left-hand side of a rule. Contains the antecedent (or conditions) that must be in working memory for the rule to execute. |
| **nonmonotonic** | Truth value changes with time. |
| **OPS83** | An artificial intelligence language that uses rules and WMEs in a forward-chaining reasoning process. |
| **pointer** | A field in a structure which indicates the address of another structure which is related in some way. |
| **real time** | Process information as fast as it is being transmitted. |
| **rewinding** | Retracing a sequence of steps. |
| **RHS** | Right-hand side of a rule. Contains the action part of a rule that will be executed if the rule is executed. Execution of the rule is under control of the inference engine. |

| | |
|---|---|
| **rule** | A software structure that contains procedural knowledge on its RHS and conditions under which to apply this knowledge on its LHS. |
| **type** | Either object or condition (as a field of an alert condition). |
| **warning** | Message displayed to the user indicating that the conditions specified in an alert have been satisfied. |
| **working memory element (WME)** | A structure that can contain data or higher level conclusions resulting from the application of relevant knowledge. WMEs are the data forms matched to the conditions on the LHSs of the rules. |

# INTRODUCTION

## PROBLEM

Aboard a naval aircraft carrier a tremendous amount of information is available on-line over communication networks and data links. Automated systems generate much of this information, necessitating automated systems for its analysis and integration. Over a period of hours or days the threat-assessment task may evolve dramatically. A computer tool to assist in threat-assessment must be able to adapt to the changing information requirements of the computer user. An expert system should be able to accommodate change by enabling the user to specify the types of information he wants processed and the categories of conclusions of which he wants to be notified.

OPS83 was used for the development of the expert system. It is a forward-chaining language, which when combined with an extensive knowledge base, can result in the derivation of far more assertions than the user can assimilate. The user must be able to tailor the program to filter and collate assertions into a more appropriate and manageable work load. This tailoring must be a run-time feature since what is appropriate is nonmonotonic (changing with time). In addition, a set of assertions generated by the OPS83 program may constitute a significant event that the program does not know about. Run-time user alerts inform the computer of the importance of this event.

An explanation is needed to instill confidence in the computer user of the validity of the conclusion, to enable him to pinpoint the source of error if a faulty conclusion is reached, and to provide him with insight into how and why the conclusion was reached.

## PURPOSE OF PROJECT

A large expert system called the Command Action Team (CAT) system was built to perform threat-assessment for a naval command ship. A main task of the project was to explore the feasibility of applying advanced artificial intelligence (AI) technology to the solution of real-time threat-assessment problems. The computer program needed to have a generalized architecture into which modules of functionality could be inserted or withdrawn with minimal sideeffects -- with rapid prototyping and test-bed architecture as design requirements. CAT also needed to be flexible to allow the user to tailor the task "on the fly" in response to the current tactical needs. A strong explanation scheme was needed to support the knowledge engineer's development needs and the user's needs for conclusion justification.

## SCOPE OF REPORT

The scope of this report is defined by its attempt to provide a detailed description of a particular alerting process and explanation processes in general. Its intended audience consists of software engineers interested in the design considerations for developing alerting and explanation procedures and for providing automated knowledge acquisition, and users or future developers of the CAT system interested in a detailed description of these processes.

# DESIGN CONSIDERATIONS

## OPS83

OPS83 is an artificial intelligence computer programming language. It uses forward-chaining inferencing, meaning that it progresses from its input data to its conclusions. (This is in contrast to a backward-chaining language, which starts with a final goal and seeks to confirm this goal given its input.) Forward chaining tends to be very responsive to its input and favors rapid switches between different paths of reasoning. Within an OPS83 program are pieces of code called rules. These rules contain knowledge from which an assertion can be made (or a conclusion drawn) given a specified set of conditions. Rules cannot be modified at run time. Thus once the program is written and put in the hands of the user, the tasks the program can perform and the conclusions it can draw are fixed for a set of input data.

## EXPLANATION PROCESS

Assertions can be derived by using many methods. The method chosen depends on (1) the algorithms that must be used, (2) the relative importance of the speed of computation versus the clarity of reasoning, and (3) whether an explanation must be provided in user-friendly terms. This report covers the constraints imposed on the reasoning method that results from the need to explain conclusions to the user.

CAT is designed to perform nonmonotonic reasoning. To do this, it must keep track of the data elements used to help derive another data element. If the original data element is found no longer to be true, it is removed from working memory. The truth of any information derived from this original data element is questionable. Unless it has independent support from other data, the derived information is also removed.

To support this backtracking process, a network of pointers is needed to indicate parent-child relationships. This same network, which can be traversed for explanation purposes, must record two types of information: factual data that exist in working memory

and knowledge that is encoded in the rules. Thus explaining the origin of an assertion involves recapitulating the rule knowledge and data elements.

The explanation scheme can either follow the reasoning path the program used to generate the conclusion or present some other reasoning sequence that could have resulted in the same conclusion. Following the program's reasoning has the advantage of needing only one reasoning sequence to achieve the conclusion. It has the disadvantage that this reasoning sequence must make sense to both the user and the knowledge engineer.

Having a separate path for idea development and explanation also has its strengths and weaknesses. It is useful when the reasoning process is so complex or abstract that to use it in the explanation would not be meaningful to a typical user. Its main disadvantage is that for every method of deriving a conclusion, there is a separate method explaining that conclusion. Also, since an assertion could be arrived at by a number of different paths, the conclusion must contain a trace of the path actually used. The explanation then retraces (or rewinds) this path in order to elaborate on the conclusions to the user.

## APPROACH

### ALERTS

#### Requirements

An alerting mechanism is needed to allow the user to configure the program at run time. An alert is a way for the user to specify a set of data elements in working memory (WMEs) as assertions that add up to a tactically significant event. The tactical significance of a set of assertions is temporal. Interest in this event also varies among the different personnel of a command. The user of the program interested in ASW (antisubmarine warfare), for example, would have a different set of alerts than the planner of an air exercise two days in the future. For a program to meet both these needs under all conditions, without having alerts set at run time, would require an astronomical number of rules. The program would either choke from the magnitude of the task or flood the user with assertions most of which would be inappropriate to his current mission.

An alert shares many traits with rules. Both alerts and rules have an antecedent (a precursor set of conditions) that, when present in working memory, triggers the execution of the alert or rule. The output of the alert is a warning displayed to the user. The output of a rule typically is changes (additions or removals) to working memory. Two major differences are: (1) Alerts are built by the user and loaded at run time. Thus they can be added or removed, depending on the tactical situation or the individual needs of the user. (2) Rules are

always active, "scanning" working memory for their antecedent conditions. If these rules address tactical areas that are not of interest to the current user, they prevent computational resources from being used for more appropriate tasks. Rules have the advantage over alerts in that they can be much more efficiently matched to WMEs than alerts. Thus if a set of assertions is of more than occasional interest, it should be encoded as a rule.

Because alerts have such a similar structure to rules, it is easy to convert the knowledge resident in alerts to rules. When users add alerts of general interest to most users, programmers can readily code the information into rules and compile this knowledge into CAT. This sequence of actions -- from developing alerts to encoding the alert contents into rules -- is a means for achieving knowledge acquisition.

### Usage

From the four information sources -- data link, static data in data bases or rules, assertions generated by CAT, and user input -- a large information pool is formed. The user would be unable to assimilate nor would he desire to see all the information in CAT. Alerts are used to allow the user to specify the events for which he wants to be notified. Alerts can be thought of as a data base query of the form "Warn me if these conditions ever exist in working memory (current data base)." Figure 1 shows the progression from the user setting an alert to developing explanation text for warnings resulting from the user's alert.

When developing a tool that can have its task modified during run time, several initial decisions must be made. The ability to perform the task can be hard-coded as rules, which are quicker and more efficient than alerts, but are active all the time. The user is unable to turn rules off if he is not interested in their utility. The decision as to what should be encoded as rules and what should be reserved for the user to add as an alert should be based on the following factors:

- How often and extensive the modifications will be.
- How rapidly the system should be able to incorporate the changes.
- How much of the total computational power will be devoted to examining alerts.

The user tailors the system to his needs by setting alerts. Alerts allow him to dictate what information should be displayed to the user. An alert consists of a set of conditions that must be present in the information pool at the same time. The intent is that each condition by itself is not of interest, but in combination they have some tactical significance in the current setting.

User defines an alert that
specifies events or information
for which he wants to be notified

This alert with its constituent set of
conditions is translated into an alert condition net

Pattern matching occurs to
satisfy each condition of the alert

When all conditions for the alert have
been met, a warning is issued

The alert condition net is traced and the
information needed for an explanation is extracted

The explanation is packed into a structure
called "Canned Text" that is printed
to the user if he asks for an explanation

**Figure 1. Path from an alert to an explanation.**

## Composition (From the User's Perspective)

Alerts deal with objects and attributes about objects. (Objects -- also known as platforms or tracks -- are such things as ships, airplanes, submarines, and land bases.) Figure 2 is an example of an alert for the following situation: Two platforms of the same category within range to intercept and with a cpa range <= 200 nmi. The title of the alert should encapsulate the main idea of the alert, as was done for the example in Fig. 2. The title is not parsed out in any way to decide what information CAT should be looking for. Rather, it should be a phrase that portrays to the user the main idea of the alert.

Two platforms of the same category within range to intercept and with a cpa range <= 200 nmi

| Platform A: Any object | Platform B: Any object |
|---|---|
| cpa range <= 200 nmi | cpa range <= 200 nmi |
| range <= its maximum range | range <= its maximum range |
| | category = category of Platform A |

**Figure 2. Sample alert.**

In analyzing this alert, CAT would divide it into eight components:

(1) Platform A has a cpa range <= 200 nmi

(2) Platform A has a maximum range

(3) Platform A has a range <= its maximum range

(4) Platform A has a category

(5) Platform B has a cpa range

(6) Platform B has a maximum range

(7) Platform B has a range <= its maximum range

(8) Platform B has a category = the category of Platform A

An occurrance of any one of these components has negligible significance on its own, but in combination they present a threat.

### Structure (From the Programmer's Perspective)

An alert consists of three types of WMEs:

(1) An "alert". This alert has a unique address that other WMEs can point to. It also contains a condition field that says how many objects the alert refers to.

(2) An "alert condition of type object" for each object specified in the alert. The object has a parent field that points to the alert WME. The object also has a unique address that other WMEs can point to and a condition number field that specifies which of the objects of the alert this object is.

(3) An "alert condition of type condition" for each condition of each object. The parent field in the condition points up to the object this condition refers to. The condition number indicates which of the object's conditions this condition is. It has a predicate field that contains the attribute that the user specified (such as category, range, speed). There are also various fields with the magnitude of the attribute the user specified (see Fig. 3 for examples). The comparison type field specifies whether this is a within- or between-object comparison. The comparison node field contains the address of the condition that is used for the comparison.

### Pattern Matching

CAT seeks to satisfy an alert condition by condition. When an assertion is found that matches a condition, several WMEs are made (see Fig. 4). One is an alert condition of type object occurrence. This WME points to the alert condition of type object via the connection

list[2] field. An alert condition of type occurrence is also made that points to the address of the satisfying assertion via the connection list[1] field. It points to the object occurrence via the object occurrence field. It also points to the condition of the alert that this occurrence satisfies. The object occurrence also points down to this Nth occurrence via the subtree list[Nth] field. Note the the "[ ]" symbology indicates that this field is an array, and the number inside the "[ ]" specifies which element in the array is being referred to.

If the object matches additional conditions of the alert, more occurrences are made until all the conditions for one alert object have been satisfied by an object in data memory. When this occurs, the status field in the object occurrence is set to satisfied. When there is an object occurrence for each object in the alert, the alert is satisfied, which is indicated by making an alert condition of type "alert occurrence." This alert occurrence lists the object occurrences in the subtree list array. Once the alert occurrence is made, a warning is issued to the user.

This network of pointers serves three purposes: First, as assertions used to satisfy conditions are removed, corresponding alert-related pointers are removed. The alert condition that was instantiated due to this condition becomes uninstantiated. Thus concurrence between all the conditions is assured. Second, checking for a complete set of pointers for an alert is how CAT knows whether the alert is satisfied. Third, this pointer network can be traced to explain why an alert was satisfied and a warning issued. Additionally this net could be used to explain why an alert was not satisfied. Tracing the net would reveal tthose alert conditions that had not been satisfied.

Figure 3 shows the alert condition network that would be required to capture the information to satisfy the sample alert. The completed alert condition-assertion net that would be made for the example in Fig. 2 is shown in Fig. 4.

## WARNINGS

When all the conditions of the alert are satisfied, a warning is made and displayed to the user. Figure 5 shows the warning that would result from the sample alert. The warning is a terse statement containing the title of the alert (it should be a descriptive phrase such as "Two platforms of the same category within range to intercept and with a cpa range <= 200 nmi.") This title in no way specifies the actual conditions of the alert, but rather describes the alert. The warning also displays the platforms used to satisfy the conditions. The warnings are numbered sequentially -- the number being a useful designation of the warning that should be explained.

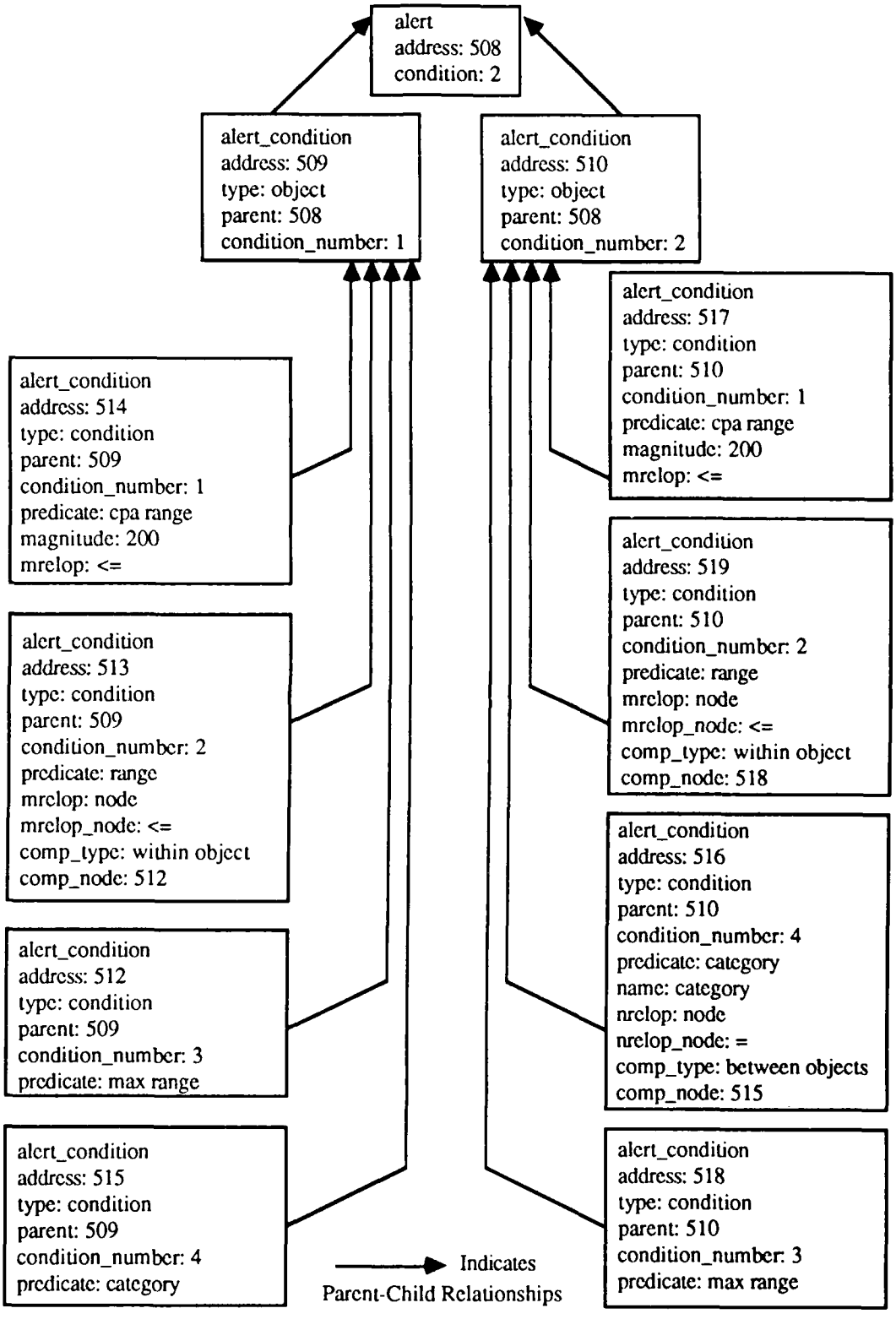



**Figure 3. Alert and alert condition network.**

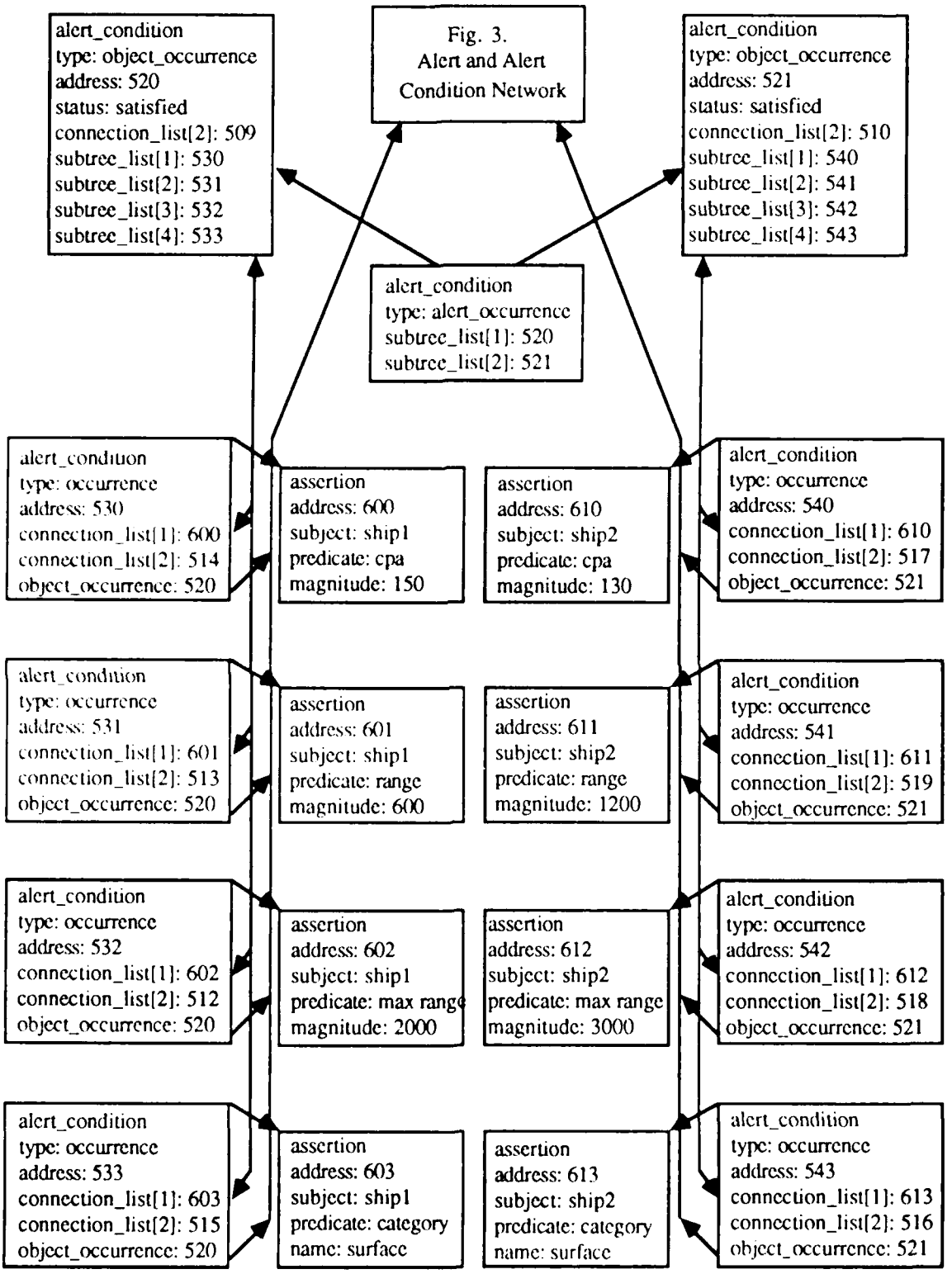



**Figure 4. Satisfied alert condition network.**

251200z Dec86 Warning #1: Alert Satisfied
Two platforms of the same category within range
to intercept and with a cpa range <= 200 nmi.
Criteria for alert satisfied by the following
platforms: ship1, ship2.

**Figure 5. Sample warning.**

## EXPLANATIONS

### Requirements

There are three main requirements which the explanation program must meet: (1) it must be able to justify the origin of the assertions used to satisfy the alert conditions, (2) it must explain how all the conditions of the alert were actualized in the data, and (3) it must assess the significance of an event in light of other events. Only the first two items are addressed further in this paper.

Justifying assertions involves outlining how the rule logic was applied to externally originating data. This mode of explanation is of interest to the knowledge engineer and programmer seeking to validate the inner workings of the program. Users of the first prototypes of the CAT program indicated that they were not interested in explanation at this depth.

Through use, CAT users become familiar with alert structures as their means of querying the data base and specifying tactically significant events. Most of CAT's output is in the form of warnings that are a direct result of alerts specified by the user. It is the warnings, then, which the users usually want to have explained. The user's intuitive approach is to present the conditions of the alert (structures familiar to the user) and show how they were realized in the WMEs.

Assertions derived by an expert system result from the firing of rules. These rules contain knowledge that can take a number of data inputs and derive an assertion from them. The firing of the rule and the generation of the assertion depend on the data being present. The general situation that a ship is in does not directly affect the rule firing. When the user asks for an explanation of the assertion, there are two levels at which the explanation can be addressed. Level one is a matter of reiterating the data and a summary of the logic used by the rule. From the knowledge engineer's perspective, this is what has actually happened. This level of explanation serves to present the specific relevant data and the knowledge in the

system.

From the user's perspective rewinding of the rule logic and antecedent data is only partially satisfactory. Perhaps more meaningful would be a level two approach, which would explain the context surrounding an assertion. Rules and alerts tend to be in groups that deal with various general situations. The presence of many assertions that deal with a certain situation would indicate the presence of that situation. The explanation of one of these assertions should indicate that this situation is present. If this assertion is present in isolation of the rest, the absence of the situation should be included in the explanation. This level of situation assessment or briefing is not implemented in CAT, but the need is recognized.

### Developing the Explanation Text

By the time a warning is given, an elaborate net of alert conditions and assertions exists. Before assertions used to satisfy conditions are removed (due to the nonmonotonic nature of CAT), the vital information that the user would want in an explanation is captured and put in the canned text. The canned text's sole purpose is for explanation. If the user requests an explanation of a warning, the canned text with that warning number is displayed. Having the information necessary for an explanation already collated provides a much faster (albeit memory intensive) explanation than tracing the extensive net when the request is made. If an explanation is never requested, this information is never used. Figure 6 shows what the canned text explanation for the sample alert would look like.

Canned text consists of a paraphrase of each alert condition followed by the assertion that was used to satisfy that condition. The format of the canned text mirrors the pattern used by CAT to satisfy the alert conditions. The explanation is intuitively natural to a user familiar with the structure of alerts. The user who has set alerts understands that a warning is issued when a certain combination of events (specified in the alert conditions) occurs.

### Tracing the Inference Net

Raw data from the electronic traffic flow are captured and input to CAT. These data are used to form an assertion and are put into a WME. An assertion is a conclusion derived by applying knowledge to data. The information for this assertion was the report from the electronic data flow. The knowledge used was the rule that extracted the information from the report and formed it into an assertion. Based on this assertion, and possibly in combination with other assertions, further assertions can be made using other knowledge (reasoning). This pattern of building assertions upon assertions results in an inference net. The sequence

From the Carl Vinson any platform
has cpa range <= 200 nmi

Note this is a paraphrase of the alert condition set by the user.

Condition(s) satisfied by:
address = 600
At 251200z Dec86
the ship1 had cpa:
range = 150 nmi
bearing = x
date/time = x
with high confidence

Note this is the actual track data which was used to satisfy the above alert condition.

From the Carl Vinson any platform
has range <= its max range

Condition(s) satisfied by:
address = 602
At 251200z Dec86
the ship1 had max range = 2000 nmi
with medium confidence

address = 601
At 251200z Dec86
the ship1 had range = 600 nmi
*with high confidence*

From the Carl Vinson any platform
has max range

Condition(s) satisfied by:
address = 602
At 251200z Dec86
the ship1 had max range = 2000 nmi
with medium confidence

From the Carl Vinson any platform
has category

Condition(s) satisfied by:
address = 603
At 251200z Dec86
the ship1 was classified as category surface
with high confidence

(Continued)

**Figure 6. Canned text.**

From the Carl Vinson any platform
has cpa range <= 200 nmi

Condition(s) satisfied by:
address = 610
At 251200z Dec86
the ship2 had cpa:
range = 130 nmi
bearing = x
date/time = x
with low confidence

From the Carl Vinson any platform
has range <= the max range

Condition(s) satisfied by:
address = 612
At 251200z Dec86
the ship2 had max range = 3000 nmi
with medium confidence

address = 611
At 251200z Dec86
the ship2 had range = 1200 nmi
with low confidence

From the Carl Vinson any platform
has max range

Condition(s) satisfied by:
address = 612
At 251200z Dec86
the ship2 had max range = 3000 nmi
with medium confidence

From the Carl Vinson any platform
has category equal to the category of platform a

Condition(s) satisfied by:
address = 603
At 251200z Dec86
with ship1 was classified as category surface
with high confidence

address = 613
At 251200z Dec86
the ship2 was classified as category surface
with medium confidence

**Figure 6. Continued.**

of operation is --

(1) Uptake live data from the data link and static data from data bases or hardcoded in CAT.

(2) Make first-level assertion on the basis of the data.

(3) Continue to make assertions until, given the data, rule firing is exhausted.

(4) Return to number 1.

There are two further considerations that require making this sequence far more complex: truth maintenance and explanation.

Truth (belief) maintenance is a method for updating the set of believed assertions. Belief maintenance is necessitated by the nonmonotonic nature of the data. Based on the data flow, a "picture" of the world is painted (assertions are drawn from the data). Additions or updates to the data are received that cause a repainting of the world image. Old data may now be obsolete. The obsolete data and assertions based on this data must be marked or removed from the system. To achieve this, some sort of trace must be made during the initial building of the assertion network. Identification of the obsolete data involves following this trace to mark the assertions based on the obsolete information.

The second reason a trace through the assertion net is needed is to provide explanations. If the expert system is to provide any explanation to the user, it must be able to arrive at its conclusions via a logical sequence that would make sense to a human. It must leave a trace from the data to the final conclusion displayed to the user. When an explanation request is entered, the program can backtrack through the net and find the information and logic (reasoning) used to derive this assertion from the data.

During the initial stages of building CAT, explanation was very low level, operating at the level of the inference net. An assertion may have been arrived at by several (or many) different paths or sets of input. All the duplicate assertions were combined, but the pointers to how they were derived were preserved. Explanation involved a plain English paraphrase of the information contained in the assertion, followed by a paraphrase of the logic used by each rule that made the assertion. The result was too fragmented for the typical user. Knowledge engineers, however, were able to follow the pieces to assess the validity of the assertions. CAT was able to account for its assertions, an important step in testing and validation.

The current level of explanation proceeds through the alert conditions, paraphrasing the condition and the assertion that satisfied the condition. The next level of explanation gives the reason for a specified assertion. At this level CAT traces the inference net, deluging the curious user with supporting reasoning and data. The result was that most users did not ask for explanations at this deep level.

# CONCLUSIONS AND RECOMMENDATIONS

## STRENGTHS

The current method of developing explanations of warnings and assertions has several strengths:

(1) The path taken during explanation duplicates that used to derive the conclusion (warning, assertion) being explained. An explanation of an invalid assertion can reveal where CAT's error occurred.

(2) Because of (1) above, multiple pathways, one for explanation and one for derivation of the conclusion, are not needed.

(3) The explanation sequence follows the same condition-by-condition approach the user used to set the alert. Thus the explanation follows a format already familiar to the user.

The alerting method is very straightforward and sequential, despite the intricacies necessary for the implementation of the actual computer code.

## LIMITATIONS

The following sections outline limitations of the alert methodology. The items should be used as suggestions for future development, and as a cautionary note for users of this computer program as to its limitations.

### Inclusion of Referenced Conditions

For within- and between-object comparisons fairly nonspecific conditions are generated for objects in order to obtain data needed for these comparisons. This is illustrated in the sample alert dealt with in Fig. 2, 3, 4, and 6. In the sample, platform B had the condition that it was in the same category as platform A. In order to make this comparison, the category of platform A had to be known. This was done by adding a condition to platform A that it has a category. What the value for this category is does not matter. At explanation time, the user is told that there is a condition for platform A, namely, that it belongs to a category. The user did not intentionally set that condition, nor does it appear in his list of conditions for the platform. Confusion can result when the user sees the explanation including conditions that he did not think were part of the alert.

Additionally, because this condition is referenced in a within- or between-object comparison, it ends up getting paraphrased twice in the explanation. This is due to the search pattern used. What happens is that condition 1 for object 1 is paraphrased along with its assertion. Then sequentially all the remaining conditions for that object are paraphrased. The

pattern repeats itself for each of the successive objects. When a condition depends on another condition, both the satisfying assertions are presented. Thus the condition that was included to link a datum and the comparison condition is paraphrased twice during the explanation -- once because it was referenced and once due to its position in the object-condition array.

### Time Window Concurrency

The current implementation does not check that the conditions of the alert are all true at the same time, only that they are all in working memory at the same time. This is much more than a semantic difference. Expert systems often keep historical data in memory for purposes of calculating trends. Thus information that was true at one point but is no longer valid is still kept in memory. Generally the user expects the conditions of the alert all to be true at the same time for the assertions to culminate in a significant event. A simple example is used to illustrate this. Assume an alert for a multiplatform hostile force within a certain surveillance area. Assume one hostile platform enters the area and then leaves. The expert system stores the closest point of approach for historical reference purposes. Later a hostile platform from a different nation enters the surveillance area. The alert would trigger a warning based on the two platforms even though they satisfied the conditions at different times. This problem can be alleviated by saying that all assertions matching alert conditions must be inferred within a certain time window. This would solve the problem in the example above, where the range assertions were made at different times. If, however, the ranges occurred near the same time but the nationality of the first track was inferred much earlier (when it first entered the track data base), the alert would not fire.

A more complex alerting mechanism would recognize what types of data are monotonic and what types are not. If, for example, one alert condition specifies what the nationality of a track must be while another condition specifies a range, a direct time window check for the concurrency of these two traits would be pointless. Nationality is generally a monotonic trait and should be considered valid until evidence dictates otherwise. Ranges of moving tracks, on the other hand, are nonmonotonic and a time-window check is appropriate. Determination of time windows can be an involved process since, for example, ranges for airplanes become outdated more quickly than ships.

### Multiple Triggering of Alert by One Track

Another limitation is one track firing an alert multiple times. Assume, for example, an alert on two objects. If there are five tracks that satisfy both objects, a warning will be triggered on

track 1 and 2, track 1 and 3... track 2 and 3, track 2 and 4, etc.

### Determining Reasons for Unsatisfied Alerts

It would be advantageous to be able to ask why an alert did not fire. To be able to trace forward down the net to see at what point it stops would allow the program to say what attribute a track lacked that kept it from satisfying an alert.

### Comparisons of Different Attributes

Another limitation of the current implementation of this alerting approach is that it does not allow for comparisons of different attributes on different tracks. For example, the user cannot set an alert comparing the "speed" of platform A with the "maximum speed" of platform B. With some enhancement, the current architecture could support this type of comparison.

### Algebraic Operators

Simple algebraic operators between conditions are not possible. For example, the user cannot check if the range of an object from its starting point is greater than half its maximum range (this would indicate that either it must refuel or that it is not making a round-trip voyage).

### Backtracking to Generate Assertions Specified in Alerts

Currently CAT's inferencing rules are independent of the alerting mechanism. The expert system is always actively trying to satisfy the rule antecedent with WMEs. When the rule's antecedent is present, it may fire, causing execution of its right-hand side (RHS). Executing the rule may produce an assertion that is not contained in any of the alert conditions and thus is wasted computation (provided this assertion is not needed for some other task).

A technique worth examination would be to build a mechanism that looks at conditions specified in alerts and turns on all the rules that try to derive those conditions. Many of the remaining rules could be turned off, greatly easing the computational burden.

### Assessing Significance of an Alert or Event

There are several more levels of explanation that could be performed. The first would be to analyze the alert that has been triggered, and based on the overall current tactical situation, deduce and explain the relevance of this satisfied alert. This level of explanation would be much more complex than what is currently implemented. It would be appealing because it

gets away from the fragmentary method used now. For a user who is neither familiar with setting alerts nor thinks in the fragmented perspective used by alerts, this would be a far smoother approach. It would collate the fragments and piece together an understanding of why a user would be interested in this combination of events. This intelligent interpretation of the significance of a warning would be complex. The mechanism would have to be cognizant of the activity of the host vessel and the other events of significance.

### Summary of Limitations

The purpose of an explanation is to fill in the details underlying major events (the major events being specified in alerts, and the details being the assertions). Untended, CAT will issue warnings each time the conditions in an alert occur. What results is a stream of warnings, each an individual entity, with nothing cohesive tying it in to other warnings being issued, nor to events occurring that are not being issued as warnings. An explanation allows the user to probe the knowledge and data in the system to find out some specific details -- the real goal is to coalesce this information into a summary of the current threats.

END

DATE

FILMED

5-88

DTIC